Text Classification: A Sequential Reading Approach

Gabriel Dulac-Arnold, Ludovic Denoyer, and Patrick Gallinari

University Pierre et Marie Curie - UPMC, LIP6
Case 169 - 4 Place Jussieu - 75005 PARIS - FRANCE
firstname.lastname@lip6.fr

Abstract. We propose to model the text classification process as a sequential decision process. In this process, an agent learns to classify documents into topics while reading the document sentences sequentially and learns to stop as soon as enough information was read for deciding. The proposed algorithm is based on a modelisation of Text Classification as a Markov Decision Process and learns by using Reinforcement Learning. Experiments on four different classical mono-label corpora show that the proposed approach performs comparably to classical SVM approaches for large training sets, and better for small training sets. In addition, the model automatically adapts its reading process to the quantity of training information provided.

1 Introduction

Text Classification (TC) is the act of taking a set of labeled text documents, learning a correlation between a document's contents and its corresponding labels, and then predicting the labels of a set of unlabeled test documents as best as possible. TC has been studied extensively, and is one of the older specialties of Information Retrieval. Classical statistical TC approaches are based on well-known machine learning models such as generative models — Naive Bayes for example [1][2] — or discriminant models such as Support Vector Machines [3]. They mainly consider the *bag of words* representation of a document (where the order of the words or sentences is lost) and try to compute a category score by looking at the entire document content. *Linear* SVMs in particular — especially for multi-label classification with many binary SVMs — have been shown to work particularly well [4]. Some major drawbacks to these *global* methods have been identified in the literature:

- These methods take into consideration a document's entire word set in order to decide to which categories it belongs. The underlying assumption is that the category information is homogeneously dispatched inside the document. This is well suited for corpora where documents are short, with little noise, so that global word frequencies can easily be correlated to topics. However, these methods will not be well suited in predicting the categories of large documents where the topic information is concentrated in only a few sentences.

– Additionally, for these methods to be applicable, the entire document must be known at the time of classification. In cases where there is a cost associated with acquiring the textual information, methods that consider the entire document cannot be efficiently or reliably applied as we do not know at what point their classification decision is well-informed while considering only a subset of the document.

Considering these drawbacks, some attempts have been made to use the sequential nature of these documents for TC and similar problems such as passage classification. The earliest models developed especially for sequence processing extend Naive Bayes with Hidden Markov Models. Denoyer et al. [5] propose an original model which aims at modeling a document as a sequence of irrelevant and relevant sections relative to a particular topic. In [6], the authors propose a model based on recurrent Neural Networks for document routing. Other approaches have proposed to extend the use of linear SVMs to sequential data, mainly through the use of string kernels [7]. Finally, sequential models have been used for Information Extraction [8,9], passage classification [10,11], or the development of search engines [12,13].

We propose a new model for Text Classification that is less affected by the aforementioned issues. Our approach models an agent that sequentially reads a text document while concurrently deciding to assign topic labels. This is modeled as a sequential process whose goal is to classify a document by focusing on its relevant sentences. The proposed model learns not only to classify a document into one or many classes, but also *when* to label, and when to stop reading the document. This last point is very important because it means that the systems is able to learn to label a document with the correct categories as soon as possible, without reading the entire text.

The contributions of this paper are three-fold:

1. We propose a new type of sequential model for text classification based on the idea of sequentially reading sentences and assigning topics to a document.
2. Additionally, we propose an algorithm using Reinforcement Learning that learns to focus on relevant sentences in the document. This algorithm also learns when to stop reading a document so that the document is classified as soon as possible. This characteristic can be useful for documents where sentence acquisition is expensive, such as large Web documents or conversational documents.
3. We show that on popular text classification corpora our model outperforms classical TC methods for small training sets and is equivalent to a baseline SVM for larger training sets while only reading a small portion of the documents. The model also shows its ability to classify by reading only a few sentences when the classification problem is easy (large training sets) and to learn to read more sentences when the task is harder (small training sets).

This document is organized as follows: In Section 2, we present an overview of our method. We formalize the algorithm as a Markov Decision Process in Section 3 and detail the approach for both multi-label and mono-label TC. We then present the set of experiments made on four different text corpora in Section 4.

2 Task Definition and General Principles of the Approach

Let $\mathcal{D}$ denote the set of all possible textual documents, and $\mathcal{Y}$ the set of C categories numbered from 1 to C. Each document d in $\mathcal{D}$ is associated with one or many[1] categories of $\mathcal{C}$. This label information is only known for a subset of documents $\mathcal{D}_{train} \subset \mathcal{D}$ called training documents, composed of N_{train} documents denoted $\mathcal{D}_{train} = (d_1, ..., d_{N_{train}})$. The labels of document d_i are given by a vector of scores $y^i = (y_1^i, ..., y_C^i)$. We assume that:

$$y_k^i = \begin{cases} 1 \text{ if } d_i \text{ belongs to category } k \\ 0 \text{ otherwise} \end{cases} . \qquad (1)$$

The goal of TC is to compute, for each document d in $\mathcal{D}$, the corresponding score for each category. The classification function f_θ with parameters θ is thus defined as :

$$f_\theta : \begin{cases} \mathcal{D} : [0;1]^C \\ d \to y^d \end{cases} . \qquad (2)$$

Learning the classifier consists in finding an optimal parameterization θ^* that reduces the mean loss such that:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} L(f_\theta(d_i), y^{d_i}), \qquad (3)$$

where L is a loss function proportional to the classification error of $f_\theta(d_i)$.

2.1 Overview of the approach

This section aims to provide an intuitive overview of our approach. The ideas presented here are formally presented in Section 3, and will only be described in a cursory manner below.

Inference We propose to model the process of text classification as a sequential decision process. In this process, our classifier reads a document sentence-by-sentence and can decide — at each step of the reading process — if the document belongs to one of the possible categories. This classifier can also chose to stop reading the document once it considers that the document has been correctly categorized.

In the example described in Fig. 1, the task is to classify a document composed of 4 sentences. The documents starts off unclassified, and the classifier begins by reading the first sentence of the document. Because it considers that the first sentence does not contain enough information to reliably classify the document, the classifier decides to read the following sentence. Having now read the first two sentences, the classifier decides that it has enough information at hand to classify the document as *cocoa.*

---

[1] In this article, we consider both the mono-label classification task, where each document is associated with exactly one category, and the multi-label task where a document can be associated with several categories.
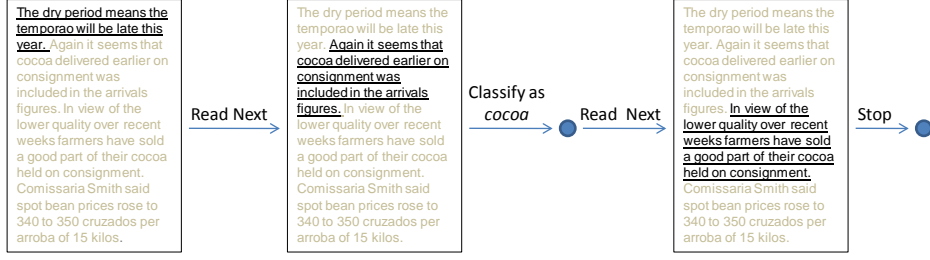
Fig. 1. Inference on a document

The classifier now reads the third sentence and — considering the information present in this sentence — decides that the reading process is finished; the document is therefore classified in the *cocoa* category.

Had the document belonged to multiple classes, the classifier could have continued to assign other categories to the document as additional information was discovered.

In this example, the model took four **actions**: *next*, *classify as cocoa*, *next* and then *stop*. The choice of each action was entirely dependent on the corresponding **state** of the reading process. The choice of actions given the state, such as those picked while classifying the example document above, is called the **policy** of the classifier. This policy — denoted π — consists of a mapping of states to actions relative to a score. This score is called a Q-value — denoted $Q(s,a)$ — and reflects the worth of choosing action a during state s of the process. Using the Q-value, the inference process can be seen as a **greedy process** which, for each timestep, chooses the best action a^* defined as the action with the highest score w.r.t. $Q(s,a)$:

$$a^* = \underset{a}{\operatorname{argmax}} Q(s,a). \tag{4}$$

Training The learning process consists in computing a $Q\text{-}function^2$ which minimizes the classification loss (as in equation (3)) of the documents in the training set. The learning procedure uses a monte-carlo approach to find a set of *good* and *bad* actions relative to each state. *Good* actions are actions that result in a small classification loss for a document. The *good* and *bad* actions are then learned by a statistical classifier, such as an SVM.

An example of the training procedure on the same example document as above is illustrated in Fig 2. To begin with, a random state of the classification process is picked. Then, for each action possible in that state, the current policy is run until it stops and the final classification loss is computed. The training algorithm then builds a set of *good* actions — the actions for which the simulation obtains the minimum loss value — and a set of remaining *bad* actions.

---

^{2} The *Q-function* is an approximation of $Q(s,a)$.

This is repeated on many different states and training documents until, at last, the model learns a classifier able to discriminate between *good* and *bad* actions relative to the current state.
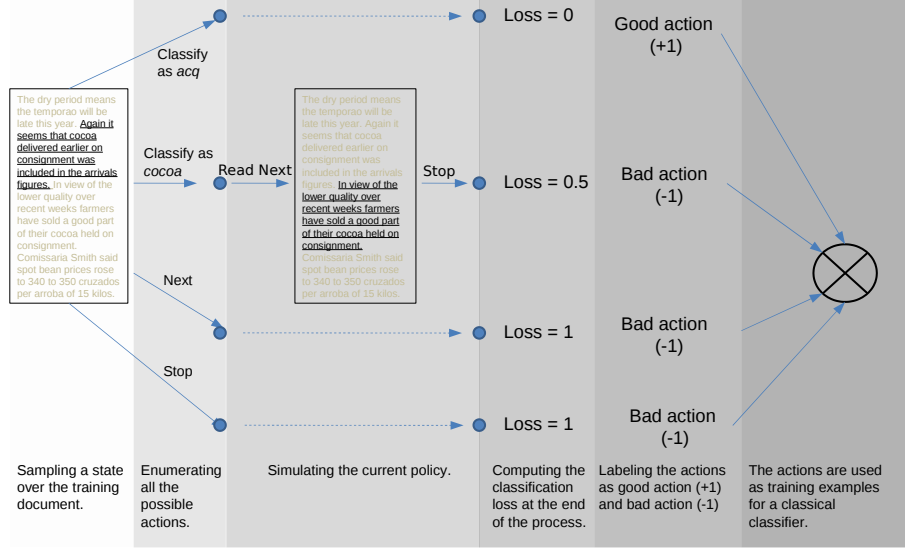

Fig. 2. Learning the sequential model. The different steps of one learning iteration are illustrated from left to right on a single training document.

2.2 Preliminaries

We have presented the principles of our approach and given an intuitive description of the inference and learning procedures. We will now formalize this algorithm as a Markov Decision Process (MDP) for which an optimal policy is found via Reinforcement Learning. Note that we will only go over notations pertinent to our approach, and that this section lacks many MDP or Reinforcement Learning definitions that are not necessary for our explanation.

Markov Decision Process A Markov Decision Process (MDP) is a mathematical formalism to model sequential decision processes. We only consider deterministic MDPs, defined by a 4-tuple: $(\mathcal{S}, \mathcal{A}, T, r)$. Here, $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the is the set of possible actions, and $T : S \times A \to S$ is the state transition function such that $T(s, a) \to s'$ (this symbolizes the system moving from state s to state s' by applying action a). The reward, $r : S \times A \to \mathbb{R}$, is a value that reflects the quality of taking action a in state s relative to the agent's ultimate goal. We will use $\mathcal{A}(s) \subseteq \mathcal{A}$ to refer to the set of possible actions available to an agent in a particular state s.

An agent interacts with the MDP by starting off in a state $s \in \mathcal{S}$. The agent then chooses an action $a \in \mathcal{A}(s)$ which moves it to a new state s' by applying

the transition $T(s, a)$. The agent obtains a reward $r(s, a)$ and then continues the process until it reaches a terminal state s_{final} where the set of possible actions is empty i.e $\mathcal{A}(s_{final}) = \emptyset$.

Reinforcement Learning Let us define $\pi : S \to A$, a stochastic policy such that $\forall a \in A(s)$, $\pi(s) = a$ with probability $P(a|s)$. The goal of RL is to find an optimal policy π^* that maximizes the cumulative reward obtain by an agent. We consider here the finite-horizon context for which the cumulative reward corresponds to the sum of the reward obtained at each step by the system, following the policy π. The goal of Reinforcement Learning is to find an optimal policy denoted π^* which maximizes the cumulative reward obtained for all the states of the process i.e.:

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{s_0 \in \mathcal{S}} \mathbb{E}_{\pi}[\sum_{t=0}^{T} r(s_t, a_t)]. \tag{5}$$

Many algorithms have been developed for finding such a policy, depending on the assumptions made on the structure of the MDP, the nature of the states (discrete or continuous), etc. In many approaches, a policy π is defined through the use of a Q-$function$ which reflects how much reward one can expect by taking action a on state s. With such a function, the policy π is defined as:

$$\pi = \operatorname*{argmax}_{a \in \mathcal{A}(s)} Q(s, a). \tag{6}$$

In such a case, the learning problem consists in finding the optimal value Q^* which results in the optimal policy π^*.

Due to the very large number of states we are facing in our approach, we consider the Approximated Reinforcement Learning context where the Q function is approximated by a parameterized function $Q_\theta(s, a)$, where θ is a set of parameters such that:

$$Q_\theta(s, a) = < \theta, \Phi(s, a) >, \tag{7}$$

where $< \cdot, \cdot >$ denotes the dot product and $\Phi(s, a)$ is a feature vector representing the state-action pair (s, a). The learning problem consists in finding the optimal parameters θ^* that results in an optimal policy:

$$\pi^* = \operatorname*{argmax}_{a \in \mathcal{A}(s)} < \theta^*, \Phi(s, a) > . \tag{8}$$

3   Text Classification as a Sequential Decision Problem

Formally, we consider that a document d is composed of a sequence of sentences such that $d = (\delta_1^d, \ldots, \delta_{n_d}^d)$, where δ_i^d is the i-th sentence of the document and n_d is the total number of sentences making up the document. Each sentence δ_i^d has a corresponding feature vector — a normalized tf-idf vector in our case — that describes its content.

3.1   MDP for Multi-Label Text Classification

We can describe our sequential decision problem using an MDP. Below, we describe the MDP for the multilabel classification problem, of which monolabel classification is just a specific instance:

- Each state s is a triplet $(d, p, \hat{y})$ such that:
    - d is the document the agent is currently reading.
    - $p \in [1, n_d]$ corresponds to the current sentence being read; this implies that δ_1^d to δ_{p-1}^d have already been read.
    - $\hat{y}$ is the set of currently assigned categories — categories previously assigned by the agent during the current reading process — where $\hat{y}_k = 1$ iff the document has been assigned to category k during the reading process, 0 otherwise.
- The set of actions $\mathcal{A}(s)$ is composed of:
    - One or many **classification** actions denoted *classify as k* for each category k where $\hat{y}_k = 0$. These actions correspond to assigning document d to category k.
    - A **next sentence** action denoted *next* which corresponds to reading the next sentence of the document.
    - A **stop action** denoted *stop* which corresponds to finishing the reading process.
- The set of transitions $T(s, a)$ act such that:
    - $T(s, \textit{classify as k})$ sets $\hat{y}_k \leftarrow 1$.
    - $T(s, \textit{next})$ sets $p \leftarrow p + 1$.
    - $T(s, \textit{stop})$ halts the decision process.
- The reward $r(s, a)$ is defined as:

$$r(s, a) = \begin{cases} F_1(y, \hat{y}) & \text{if } a \text{ is a } \textit{stop} \text{ action} \\ 0 & \text{otherwise} \end{cases}, \tag{9}$$

where y is the real vector of categories for d and $\hat{y}$ is the predicted vector of categories at the end of the classification process. The F_1 score of a single document is defined as:

$$F_1(y, \hat{y}) = 2 \cdot \frac{p(y, \hat{y}) \cdot r(y, \hat{y})}{p(y, \hat{y}) + r(y, \hat{y})} \tag{10}$$

$$\text{with} \tag{11}$$

$$p(y, \hat{y}) = \sum_{k=0}^{C} \mathbb{1}(\hat{y}_k = y_k) / \sum_{k=0}^{C} \hat{y}_k \text{ and } r(y, \hat{y}) = \sum_{k=0}^{C} \mathbb{1}(\hat{y}_k = y_k) / \sum_{k=0}^{C} y_k. \tag{12}$$

MDP for Mono-Label Text Classification   In mono-label classification, we restrict the set of possible actions. The *classify as k* action leads to a stopping state such that $A(s) = \{stop\}$. This brings the episode to an end after the attribution of a single label. Note that in the case of a mono-label system — where only one category can be assigned to a document — the reward corresponds to a classical accuracy measure: 1 if the chosen category is correct, and 0 otherwise.

3.2 Features over states

We must now define a feature function which provides a vector representation of a state-action pair (s, a). The purpose of this vector is to be able to present (s, a) to a statistical classifier to know whether it is *good* or *bad*. Comparing the scores of various (s, a) pairs for a given state s allows us to choose the best action for that state.

Classical text classification methods only represent documents by a global — and usually tf-idf weighted — vector. We choose, however, to include not only a global representation of the sentences read so far, but also a local component corresponding to the most recently read sentence. Moreover, while in state s, a document may have been already assigned to a set of categories; the global feature vector $\Phi(s, a)$ must describe all this information. The vector representation of a state s is thus defined as $\Phi(s)$:

$$\Phi(s) = \left(\frac{\sum_{i=1}^{p} \delta_i^d}{p} \ \ \delta_p^d \ \ \hat{y_0} \dots \hat{y_C} \right). \tag{13}$$

$\Phi(s)$ is the concatenation of a set of sub-vectors describing: the mean of the feature vectors of the read sentences, the feature vector of the last sentence, and the set of already assigned categories.

In order to include the action information, we use the block-vector trick introduced by [14] which consists in projecting $\Phi(s)$ into a higher dimensional space such that:

$$\Phi(s, a) = (0 \dots \phi(s) \dots 0). \tag{14}$$

The position of $\Phi(s)$ inside the global vector $\Phi(s, a)$ is dependent on action a. This results in a very high dimensional space which is easier to classify in with a linear model.

3.3 Learning and Finding the optimal classification policy

In order to find the best classification policy, we used a recent Reinforcement Learning algorithm called *Approximate Policy Iteration with Rollouts*. In brief, this method uses a monte-carlo approach to evaluate the quality of all the possible actions amongst some random sampled states, and then learns a classifier whose goal is to discriminate between the *good* and *bad* actions relative to each state. Due to a lack of space, we do not detail the learning procedure here and refer to the paper by Lagoudakis et al [15]. An intuitive description of the procedure is given in Section 2.1.

4 Experimental Results

We have applied our model on four different popular datasets: three mono-label and one multi-label. All datasets were pre-processed in the same manner: all punctuation except for periods were removed, SMART stop-words[16] and words less than three characters long were removed, and all words were stemmed with

Porter stemming. Baseline evaluations were performed with libSVM[17] on normalized tf-idf weighted vectorial representations of each document as has been done in [3]. Published performance benchmarks can be found in [18] and [19]. The datasets are:

- The Reuters-21578[3] dataset which provides two corpora:
    - The **Reuters8** corpus, a mono-label corpus composed of the 8 largest categories.
    - The **Reuters10** corpus, a multi-label corpus composed of the 10 largest categories.
- The WebKB[4][20] dataset is a mono-label corpus composed of Web pages dispatched into 4 different categories.
- The 20 Newsgroups[5] (20NG) dataset is a mono-label corpus of news composed of 20 classes.

Corpus	Nb of documents	Nb of categories	Nb of sentences by doc.	Task
R8	7678	8	8.19	Mono-label
R10	12 902	10	9.13	Multi-label
Newsgroup	18 846	20	22.34	Mono-label
WebKB	4 177	4	42.36	Mono-label

Table 1. Corpora statistics.

4.1 Evaluation Protocol

Many classification systems are *soft classification* systems that compute a score for each possible category-document pair. Our system is a *hard classification* system that assigns a document to one or many categories, with a score of either 1 or 0. The evaluation measures used in the litterature, such as the *breakeven* point, are not suitable for *hard classification* models and cannot be used to evaluate and compare our approach with other methods. We have therefore chosen to use the *micro-F1* and *macro-F1* measures. These measures correspond to a classical F_1 score computed for each category and averaged over the categories. The *macro-F1* measure does not take into account the size of the categories, whereas the *micro-F1* average is weighted by the size of each category. We averaged the different models' performances on various train/test splits that were randomly generated from the original dataset. We used the same approach both for evaluating our approach and the baseline approaches to be able to compare our results properly. For each training size, the performance of the models were averaged over 5 runs. The hyper-parameters of the SVM and the hyper-parameters of the RL-based approach were manually tuned. What we present here are the best results obtained over the various parameter choices we tested. For the RL approach, each policy was learned on 10,000 randomly generated states, with 1 rollout per state, using a random initial policy. It is important to note that, in a practical sense, the RL method is not much more complicated to tune than a classical SVM since it is rather robust regarding the values of the hyper-parameters.

---

[3] http://web.ist.utl.pt/%7Eacardoso/datasets/
[4] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/
[5] http://people.csail.mit.edu/jrennie/20Newsgroups/

4.2   Experimental Results

Our performance figures use SVM to denote baseline Support Vector Machine performance, and STC (Sequential Text Classification) to denote our approach. In the case of the mono-label experiments (Figure 3 and 4-left), performance of both the SVM method and our method are comparable. It is important to note, however, that in the case of small training sizes (1%, 5%), the STC approach outperforms SVM by 1-10% depending on the corpus. For example, on the R8 dataset we can see that for both *F1* scores, STC is better by $\sim 5\%$ with a training size of 1%. This is also visible with the NewsGroup dataset, where STC is better by 10% for both metrics using a 1% training set. This shows that STC is particularly advantageous with small training sets.

The reading process' behaviour is explored in Figure 5. Here, *Reading Size* corresponds to the mean percentage of sentences read for each document[6]. We can see that *Reading Size* decreases as the training size gets bigger for mono-label corpora. This is due to the fact that the smaller training sizes are harder to learn, and therefore the agent needs more information to properly label documents. In the right-hand side of Figure 5, we can see a histogram of number of documents grouped by *Reading Size*. We notice that although there is a mean *Reading Size* of 41%, most of the documents are barely read, with a few outliers that are read until the end. The agent is clearly capable of choosing to read more or less of the document depending on its content.

In the case of multi-label classification, results are quite different. First, we see that for the R10 corpus, our model's performance is lower than the baseline on large training sets. Moreover, the multi-label model reads all the sentences of the document during the classification process. This behaviour seems normal because when dealing with multi-label documents, one cannot be sure that the remaining sentences will not contain relevant information pertaining to a particular topic. We hypothesize that the lower performances are due to the much larger action space in the multi-label problem, and the fact that we are learning a single model for all classes instead of one independent models per class.

5   Conclusions

We have presented a new model that learns to classify by sequentially reading the sentences of a document, and which labels this document as soon as it has collected enough information. This method shows some interesting properties on different datasets. Particularly in mono-label TC, the model automatically learns to read only a small part of the documents when the training set is large, and the whole documents when the training set is small. It is thus able to adapt its behaviour to the difficulty of the classification task, which results in obtaining faster systems for easier problems. The performances obtained are close to the performance of a baseline SVM model for large training sets, and better for small training sets.

---

[6] If l_i is the number of sentences in document i read during the classification process, and n_i is the total number of sentences in this document. Let N be the number of test documents, then the reading size value is $\frac{1}{N} \sum_i \frac{l_i}{n_i}$.
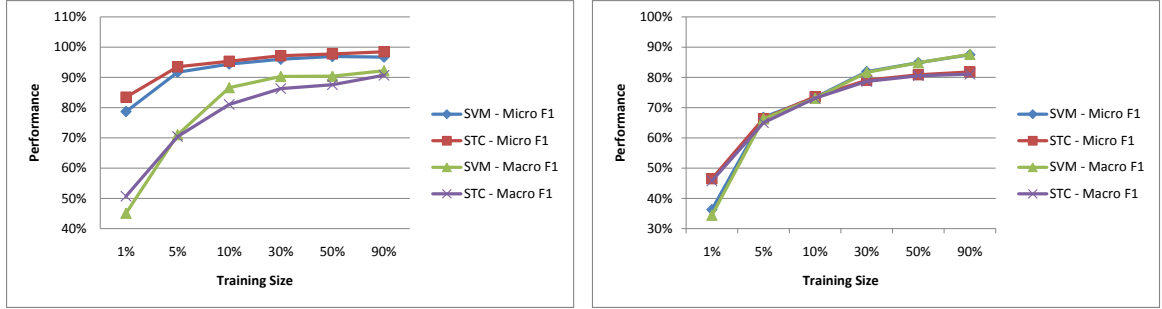
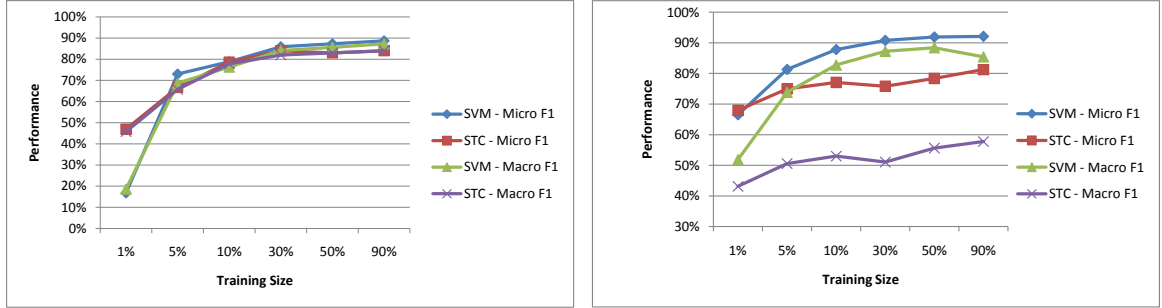**Fig. 3.** Performances over the R8 Corpus (left) and NewsGroup Corpus (right)

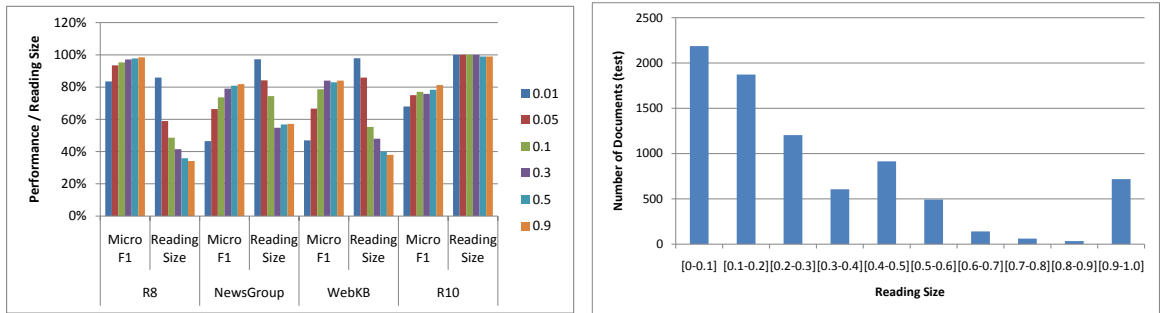**Fig. 4.** Performances over the WebKB Corpus (left) and R10 Corpus (right)

Fig. 5. Overview of the Reading Sizes for all the corpora (left). Number of documents and Reading Sizes on R8 with 30% of documents as a training set (right).

This work opens many new perspectives in the Text Classification domain. Particularly, it is possible to imagine some additional MDP actions for the classification agent allowing the agent to parse the document in a more complex manner. For example, this idea can be extended to learn to classify XML documents reading only the relevant parts.

Acknowledgments

References

1. D. Lewis and M. Ringuette, "A comparison of two learning algorithms for text categorization," *Third annual symposium on document analysis*, pp. 1–14, 1994.
2. D. Lewis, R. Schapire, J. Callan, and R, "Training algorithms for linear text classifiers," *ACM SIGIR*, pp. 120–123, 1996.
3. T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," *Machine Learning: ECML-98*, 1998.
4. S. Dumais, J. Platt, D. Heckerman, and M, "Inductive learning algorithms and representations for text categorization," *Proceedings of CIKM*, 1998.
5. L. Denoyer, H. Zaragoza, and P. Gallinari, "HMM-based passage models for document classification and ranking," in *Proceedings of ECIR-01*, 2001, pp. 126–135.
6. S. Wermter, G. Arevian, and C. Panchev, "Recurrent neural network learning for text routing," vol. 2, 1999, pp. 898 –903 vol.2.
7. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *J. Mach. Learn. Res.*, vol. 2, pp. 419–444, 2002.
8. T. R. Leek, "Information extraction using hidden markov models," 1997.
9. M.-R. Amini, H. Zaragoza, and P. Gallinari, "Learning for sequence extraction tasks," in *RIAO*, 2000, pp. 476–490.
10. M. Kaszkiel, J. Zobel, and R. Sacks-Davis, "Efficient passage ranking for document databases," *ACM Trans. Inf. Syst.*, vol. 17, no. 4, pp. 406–439, 1999.
11. J. Jiang and C. Zhai, "Extraction of coherent relevant passages using hidden markov models," *ACM Trans. Inf. Syst.*, vol. 24, no. 3, pp. 295–319, 2006.
12. D. R. H. Miller, T. Leek, and R. M. Schwartz, "Bbn at trec7: Using hidden markov models for information retrieval," in *In Proceedings of TREC-7*, 1999, pp. 133–142.
13. M. Bendersky and O. Kurland, "Utilizing passage-based language models for document retrieval," in *ECIR'08*, 2008, pp. 162–174.
14. S. Har-Peled, D. Roth, and D. Zimak, "Constraint classification: A new approach to multiclass classification," *Algorithmic Learning Theory*, pp. 1 – 11, 2002.
15. M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," *ICML*, 2003.
16. G. Salton, Ed., *The SMART Retrieval System - Experiments in Automatic Document Processing.* Englewood, Cliffs, New Jersey: Prentice Hall, 1971.
17. C.-C. Chang and C.-J. Lin, "LIBSVM: a library for SVMs," 2001.
18. F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, Mar. 2002.
19. M. A. Kumar and M. Gopal, "A comparison study on multiple binary-class SVM methods for unilabel text categorization," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1437–1444, Aug. 2010.
20. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to extract symbolic knowledge from the World Wide Web," *World*, 1998.